m

crocus

Growth towards impact

Guiding growth
in the education sector

Loading Large Datasets into the ODS/API - Reviewing Alternatives to the Data Import Tool

ed-fi ALLIANCE Prepared in collaboration with the Ed-Fi Alliance

November 1, 2022

Table of Contents

Overview

Ed-Fi implementations often require loading large amounts of data on a daily basis through the Ed-Fi API. Sometimes,when working with large datasets, data loads implemented using Ed-Fi's Data Import Tool (DIT) take longer than the refresh period to complete, rendering the use of the tool unfeasible. As an example, one implementer needed to ingest millions of records every 24 hours which is not a load meant to be handled by the DIT (which at most can handle 10's of thousands of records per day). An alternative to the DIT is needed in such cases..

Scope

This document reviews a number of ETL (Extract, Transform and Load) tools compatible with the Ed-Fi API. It includes both a survey of popular ETL tools in the market as well as a deep dive into several tools that are considered good alternatives to DIT..

ETL Tool Survey

An existing list of ETL tools was used as a starting place for the survey. A review of each was performed to get an initial idea of how good a replacement they would be to the DIT. The main areas of interest were:

- Completeness: A check to confirm whether the tool can perform all ETL operations .
- Ease of use: A subjective assessment of how easy it is to learn and work with the tool.
- Any pros and cons noted during the review.

Survey Data

Tool	Complete ETL	Ease of Use/ Installation	Pros	Cons
Talend	Extract Transform Load	Easy/Easy	- Multi-platform - Tool not required to run jobs (standalone Java apps) - Job Orchestration within tool - Open-Source - Generates code (Java, read-only) - UI	- Concurrency only on paid version
Singer	Extract ~~Transform~~ ~~Load~~	Difficult/Difficult	- Open-Source - Requires developing a custom Tap (source) for fs CSV (S3 CSV available)	- Requires developing a custom Target for Ed-Fi API
Airbyte	Extract Transform Load	Medium/Easy	- Open-Source version available - Docker image	- No connector available for EdFi API or generic REST - Connectors in Alpha state - Limited transformations
Camel	Extract Transform Load	Difficult/Difficult	- Open-Source	- Library only

Nifi	Extract Transform Load	Medium/Easy	• Open-Source • Web UI • InvokeHTTP controller for EdFi API interaction available Out-Of-Box • Concurrency Out-Of-Box	• Complex
Hevo	Extract Transform Load	Medium/Easy	• MSSQL & Postgres connector available • Python transformations • UI for transformations (beta)	• Cloud only • No REST as destination
Stitch	Extract Transform Load	Medium/Easy	• UI for transformations	• Cloud only • No REST as destination
Matillion	Extract Transform Load	Medium/Easy	• UI for transformations	• Cloud only • No REST as destination
Informatica	Extract Transform Load	Medium/Easy	• Cloud and on-prem agents • UI for transformations • REST as destination	• Connectors are paid individually
Fivetran	Extract Transform Load	Medium/Easy	• UI for transformations	• Cloud only • No REST as destination
AWS Glue	Extract Transform Load	Difficult/Easy	• UI for transformations • UI generates Python code • Databricks based	• Target and destinations limited to AWS • No REST as destination (from UI)

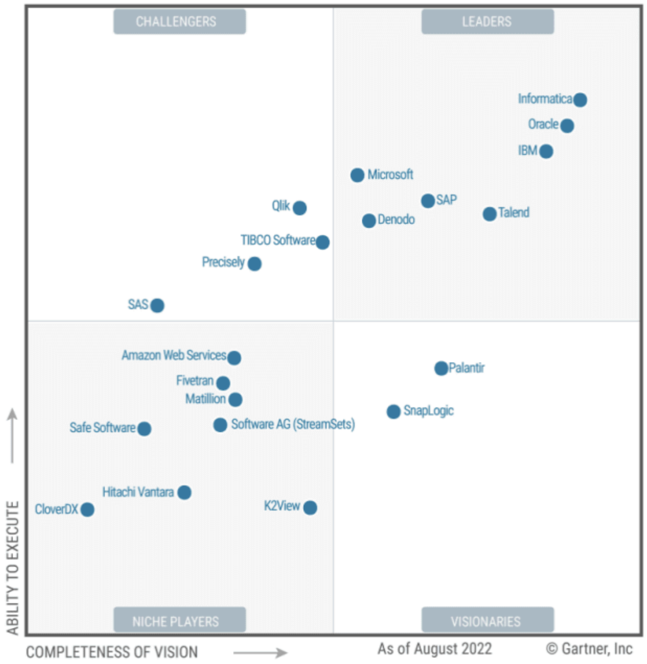
Azure Data Factory	Extract Transform Load	Medium/Easy	- UI for transformations - UI generates JSON configs - Databricks based	- REST as destination
Databricks	Extract Transform Load	Difficult/Medium	- Available thru AWS, GCP, Azure, etc. - Can use SQL	- No UI for transformation
Data Import Tool	Extract ~~Transform~~ Load	Easy/Medium	- Simple	- No transformations - Limited sources (CSV) - Poor performance (single threaded)

Survey Criteria

In order to reduce the number of tools to evaluate the following criteria was used:

- **Ease of Use:** since the DIT provides a UI and it's relatively easy to use for new users, the tool has to provide a UI.
- **Ease of Installation:** the tool has to be easy to set-up (on-prem or cloud).
- **Cost:** open source and pay-per-use were preferred over subscription based tools.
- **Completeness:** the DIT only provides the loading portion of an ETL application. Extraction, transformations and automations have to be done via scripting outside of it. Tools which provide more features were preferred.
- **Performance:** given the poor performance of the DIT with large data sets, tools were evaluated on how fast they could load records thru the Ed-Fi API. Tools which provide concurrency out of the box were preferred.
- **Popularity:** tools which have already been used in previous implementations or that are mentioned in reports such as the 2022 Gartner® Magic Quadrant™ for Data Integration Tools report (figure 1) were prioritized.

Figure 1: Magic Quadrant for Data Integration Tools



Source: Gartner (August 2022)

Survey Results

Based on the criteria identified, two on-prem tools and one cloud tool were selected to conduct a proof of concept to confirm whether they are a valid replacement for the DIT.

These are:

- On-Prem:
  - Talend Open Studio: Mentioned as a Leader by Gartner, open source version available with UI and REST component.
  - Nifi: Mentioned as a Leader by Gartner, open source version available with UI and REST component.
- Cloud:

- Azure Data Factory: Mentioned as a Niche Player by Gartner, available with UI and REST component.

Deep Dive POC

The proof of concept consisted in loading a medium sized data source (around 100.000 records) thru the Ed-Fi API into an Ed-Fi ODS.

Data Source

As data source, the NCES CCD 2020-2021 school directory available at https://nces.ed.gov/ccd/files.asp#Fiscal:2,SchoolYearId:35,Page:1 was used. This file consists of 101.662 school records covering the universe of all public elementary and secondary schools in the United States.

Data Mapping

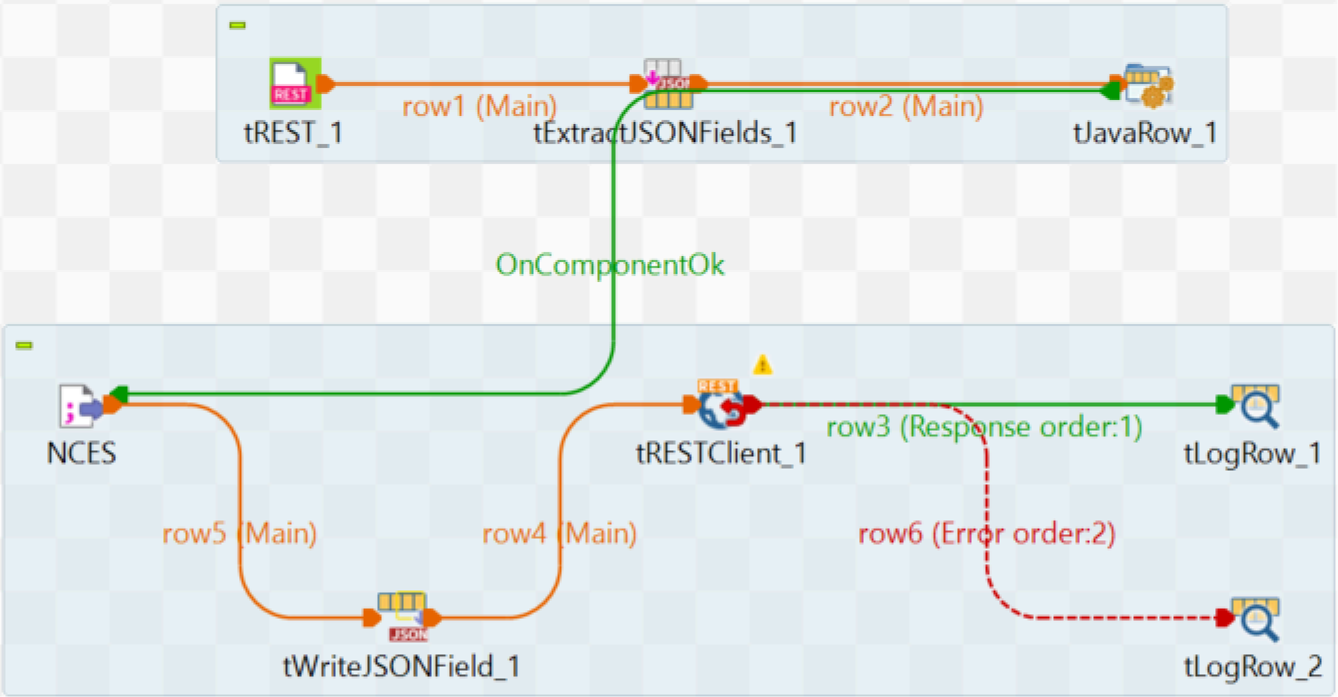
The **schools** API endpoint was mapped as follows:

Endpoint field	Data Source column / value
schoolId	SCHID
nameOfInstitution	SCH_NAME
gradeLevels	[{'gradeLevelDescriptor':'uri://ed-fi.org/GradeLevelDescriptor#Early Education'}]
educationOrganizationCategories	[{'educationOrganizationCategoryDescriptor':'uri://ed-fi.org/EducationOrganizationCategoryDescriptor#School'}]

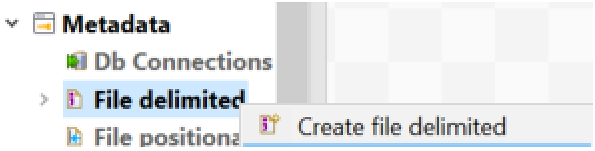
Talend Open Studio (v8.0)

This tool can be downloaded from https://www.talend.com/lp/open-studio-for-data-integration/. This being a Java tool it requires a Java environment. The required Java version is 11.

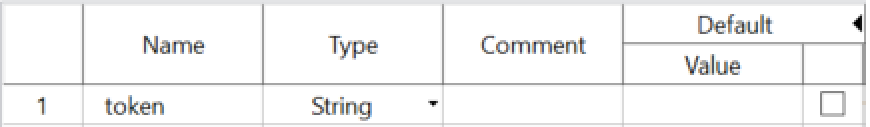
The following job was created:



In the Metadata -> File delimited option a new metadata named NCES was configured for the data source file:



In the contexts for this job a new variable was created as displayed:



	Name	Type	Comment	Default	
				Value	
1	token	String			☐

The components used in the job are described below (only parameters changed from their default values are described):

- tREST_1: this tREST is used to retrieve a bearer token from the Ed-Fi API that will be used to authorize the subsequent calls to the schools endpoint. The following parameters where configured:
  - URL: Oauth endpoint for the Ed-Fi API (e.g:"https://edfi-test/WebApi/oauth/token").
  - HTTP Method: POST
  - HTTP Headers: a new header was added with the following values:

name	value
“Content-Type”	“application/json”

  - HTTP Body:

```
"{
    'client_id': '<vendor id>',
    'client_secret': '<vendor secret>',
    'grant_type': 'client_credentials'
}"
```

- tExtractJSONFields_1: this tExtractJSONFields component is used to extract the token from the JSON return by the authentication endpoint. Its parameters were configured as follows:
  - Read By: JsonPath
  - JSON field: Body
  - Loop Jsonpath query: “$”
  - Mapping:

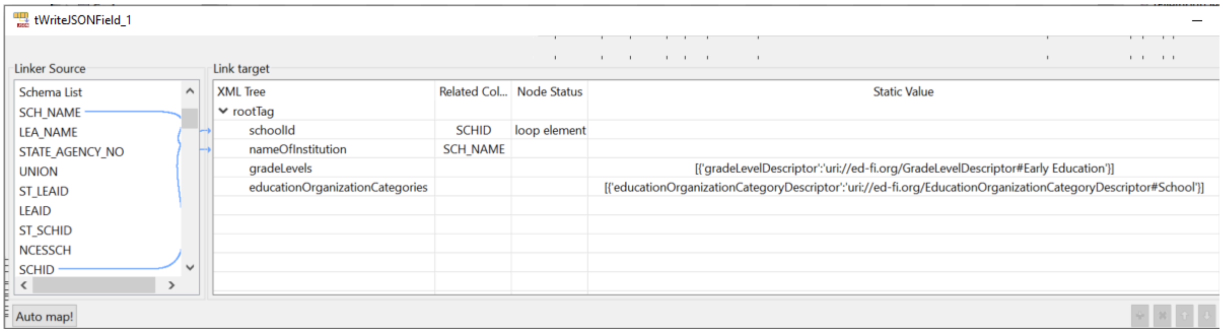
Column	Json query
access_token	“access_token”

- tJavaRow_1: a tJavaRow component used to store the access token into a context variable with the following parameters:

  - Code: note only the first line is required. The other lines just print out the access token value when running the job.

```
context.token=input_row.access_token;
java.io.PrintStream consoleOut_jr = new java.io.PrintStream(System.out);
consoleOut_jr.println(context.token);
consoleOut_jr.flush();
```

- NCES: this tFileInputDelimited component reads the data source file. It was created by drag & drop the previously configured NCES metadata file.
- tWriteJSONField_1: a tWriteJSONField configured with the following parameter:
  - Remove root node: <checked>
  - Map (double click on component):



Schema List
SCH_NAME
LEA_NAME
STATE_AGENCY_NO
UNION
ST_LEAID
LEAID
ST_SCHID
NCESSCH
SCHID

XML Tree	Related Col...	Node Status	Static Value
rootTag			
schoolId	SCHID	loop element	
nameOfInstitution	SCH_NAME		
gradeLevels			[{'gradeLevelDescriptor':'uri://ed-fi.org/GradeLevelDescriptor#Early Education'}]
educationOrganizationCategories			[{'educationOrganizationCategoryDescriptor':'uri://ed-fi.org/EducationOrganizationCategoryDescriptor#School'}]

- tRESTClient_1: a tRestClient used to POST each JSON payload to the schools API endpoint configured as follows:
  - URL: Ed-Fi API schools endpoint (e.g: "https://edfi-test/WebApi/data/v3/ed-fi/schools")
  - HTTP Method: POST
  - Content Type: JSON
  - Accept Type: Any
  - Use Authentication: <checked>
  - Authentication Type: OAuth2 Bearer
  - Bearer Token: context.token
- Two tLogRow components with default parameters used to log API responses.

This job took longer than the DIT to load the entire file on the same machine since on the free version of the tool parallelization is not enabled. As per the tool documentation: "Note that this type of parallelization is available only on the condition that you have subscribed to one of the Platform solutions or Big Data solutions.”

The tool allows building a “standalone job” (java application) that can be run without Talend Studio.

In order to send POSTs to the Ed-Fi API, the server certificate must be added to the JVM cacerts store by:

1. Export the certificate: On windows -> "Manage computer certificates" -> locate certificate -> Export (.CER file)
2. Use the keytool in the JDK to import it:

```
.\keytool.exe -importcert -trustcacerts -cacerts -file edfiods.cer -alias EdFiOds
```

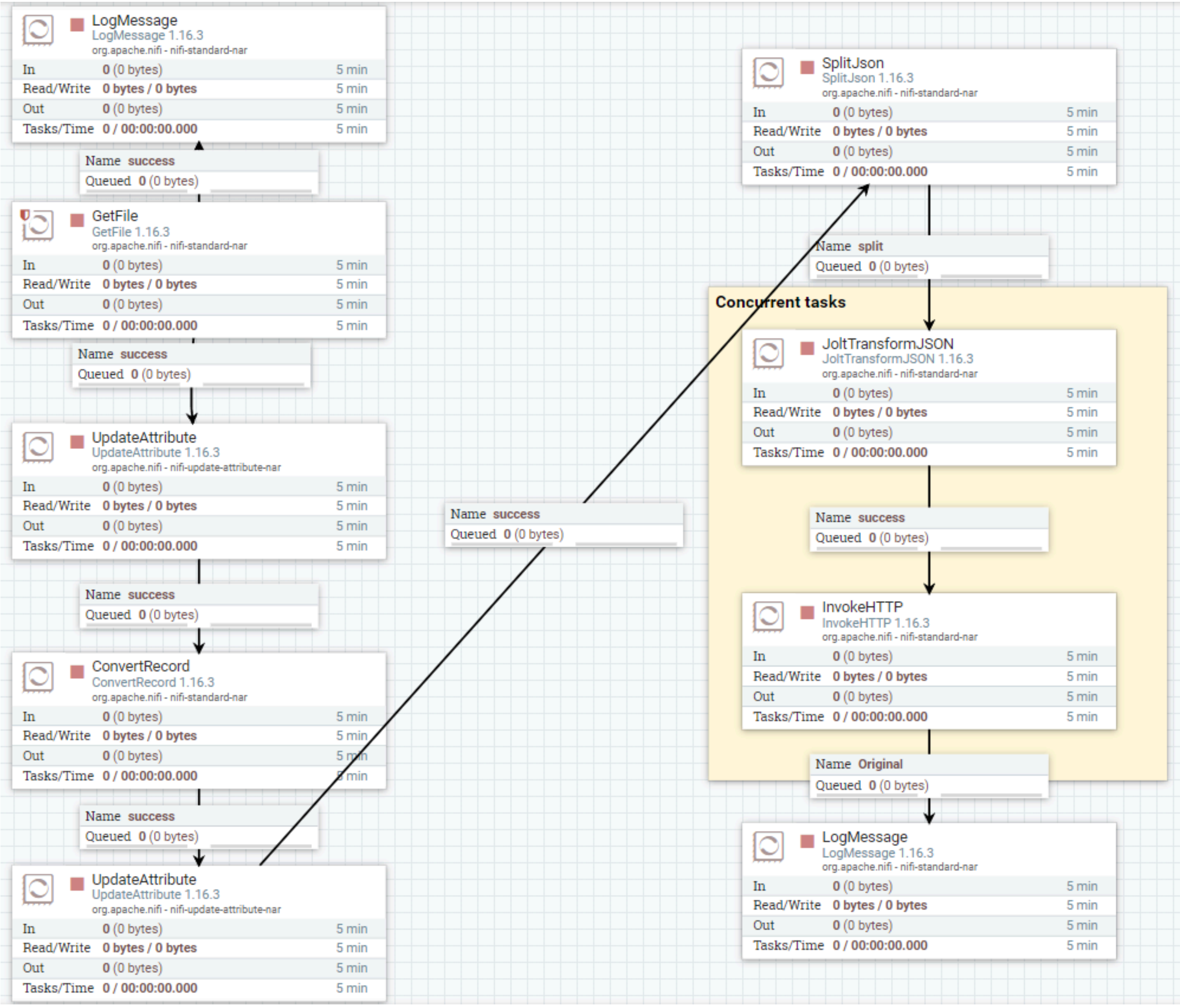
The default password for the JVM certificate store is "changeit".

Nifi (v1.16.3)

There's no installer for this tool. It's a web application running on apache. To use it:

1. Download it from https://nifi.apache.org/download.html.
2. Extract the downloaded folder.
3. Run the /bin/run-nifi.bat file from the folder where you extracted Nifi.
4. On your browser open https://localhost:8443/nifi.
5. Locate the autogenerated credentials in the /logs/nifi-app.log file by searching for "Generated Username".
6. Use the found credentials in the Nifi Log in screen.

The following flow was created:



LogMessage
LogMessage 1.16.3
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

GetFile
GetFile 1.16.3
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

UpdateAttribute
UpdateAttribute 1.16.3
org.apache.nifi - nifi-update-attribute-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

ConvertRecord
ConvertRecord 1.16.3
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

UpdateAttribute
UpdateAttribute 1.16.3
org.apache.nifi - nifi-update-attribute-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

SplitJson
SplitJson 1.16.3
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name split
Queued 0 (0 bytes)

Concurrent tasks

JoltTransformJSON
JoltTransformJSON 1.16.3
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

InvokeHTTP
InvokeHTTP 1.16.3
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name Original
Queued 0 (0 bytes)

LogMessage
LogMessage 1.16.3
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

These processors were used:

- LogMessage (upper left corner): Logs the current time. It serves to log at which time the process starts immediately after reading the data source by configuring these parameters:
    - Log message: start=${now()}
- GetFile: creates a FlowFile from the data source csv. It continually polls the input directory while the processor is running. It was configured as follows:
    - Input Directory: path to the data source file.

  - File Filter: .*\.csv

- Update Attribute: Updates the Attributes for a FlowFile by using the Attribute Expression Language and/or deletes the attributes based on a regular expression. It's used to add a new column to the file read by adding a new property to it as follows:

Property	Value
schema.name	schools

- ConvertRecord: Converts records from one data format to another using configured Record Reader and Record Write Controller Services, in this case it's used to transform the CSV rows into JSON. Its properties were configured as follows:
  - Record Reader: CSVReader
  - Record Writer: JsonRecordSetWriter
- UpdateAttribute (lower left corner):this processor is used to add another attribute to the FlowFile. It's configured by adding a new property as follows:

Property	Value
filename	$(filename).json

- SplitJson: Splits a JSON File into multiple, separate FlowFiles for an array element specified by a JsonPath expression. In this flow creates a JSON object for each CSV row by configuring the following properties:
  - JsonPath Expression: $.*
- JoltTransformJSON: Applies a list of Jolt specifications to the flowfile JSON payload. It is used to map the CSV columns into specific JSON fields and to hardcode other JSON fields. This was achieved by setting the next properties:
  - Jolt Transformation DSL: Shift
  - Jolt Specification:

```
{    "SCHID": "schoolId",
     "SCH_NAME": "nameOfInstitution",
     "": "educationOrganizationCategories[]",
     "#uri://ed-fi.org/EducationOrganizationCategoryDescriptor#School":
```

```
"educationOrganizationCategories[].educationOrganizationCategoryDescriptor",
     "": "gradeLevels[]",
     "#uri://ed-fi.org/GradeLevelDescriptor#Early Education":
"gradeLevels[].gradeLevelDescriptor" }
```

  Additionally these scheduling parameters were configured:

  - Concurrent Tasks: 20

- InvokeHTTP: An HTTP client processor which can interact with a configurable HTTP Endpoint. It is used to send the JSON payloads to the API endpoint. Its properties were configured as follows:
  - HTTP Method: POST
  - Remote URL: Ed-Fi API schools endpoint (e.g: "https://edfi-test/WebApi/data/v3/ed-fi/schools")
  - OAuth2 Access Token provider: StandardOauth2AccessTokenProvider
  - Content-Type: application/json

  Additionally these scheduling parameters were configured:

  - Concurrent Tasks: 20

- LogMessage (bottom right corner): another processor used to log the end time for the process by configuring its parameters as following:
  - Log message: end=${now()}

The following controller services were configured:

- CSVReader: Parses CSV-formatted data, returning each row in the CSV file as a separate record. It's used by the ConvertRecord processor.
- AvroSchemaRegistry: a service for registering and accessing schemas. It is used by the JsonRecordSetWriter controller service. A new property was added to it with the following:

Property	Value
schools	{"type":"record", "name":"schoolRecord", "fields":[{"name": "SCHID", "type":"long"}, {"name": "SCH_NAME", "type":"string"}]}

- JsonRecordSetWriter: Writes the results of a RecordSet as either a JSON Array or one JSON object per line. It's used by the ConvertRecord processor with the following properties:
    - Schema Write Strategy: Set 'schema.name' Attribute
    - Schema Access Strategy: Use 'Schema Name' Property
    - Schema Registry: AvroSchemaRegistry
    - Schema Name: ${schema.name}

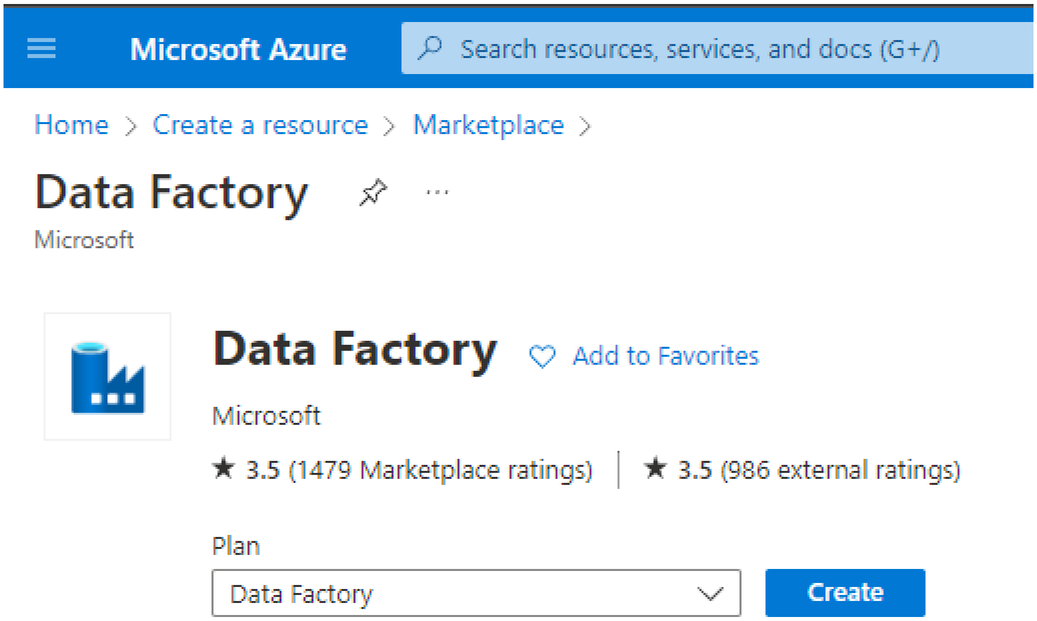
- StandardOauth2AccessTokenProvider: Provides OAuth 2.0 access tokens that can be used as Bearer authorization header in HTTP requests. Used by the InvokeHTTP controller with the following properties:
    - Authorization Server URL: Oauth endpoint for the Ed-Fi API (e.g:"https://edfi-test/WebApi/oauth/token").
    - Grant Type: Client Credentials
    - Client ID: <vendor Id>
    - Client secret: <vendor secret>

Finally, on the "Controller Settings" menu, under the "General" tab the "Maximum Timer Driven Thread Count" was set to "20".

It took this tool 6 to 10 times less to load all source records than the DIT or Talend due to its out-of-box parallelization. Notably, it's possible to build a NiFi cluster where all nodes run the same data flow increasing parallelization and reducing load times, provided the Ed-Fi API service can serve the load.

Azure Data Factory

Since this is a cloud tool there's no installation required. It is set-up by creating a "Data Factory" resource from the Azure portal:


Microsoft Azure
Search resources, services, and docs (G+/)
Home > Create a resource > Marketplace >
Data Factory
Microsoft
Data Factory
Add to Favorites
Microsoft
★ 3.5 (1479 Marketplace ratings)
★ 3.5 (986 external ratings)
Create

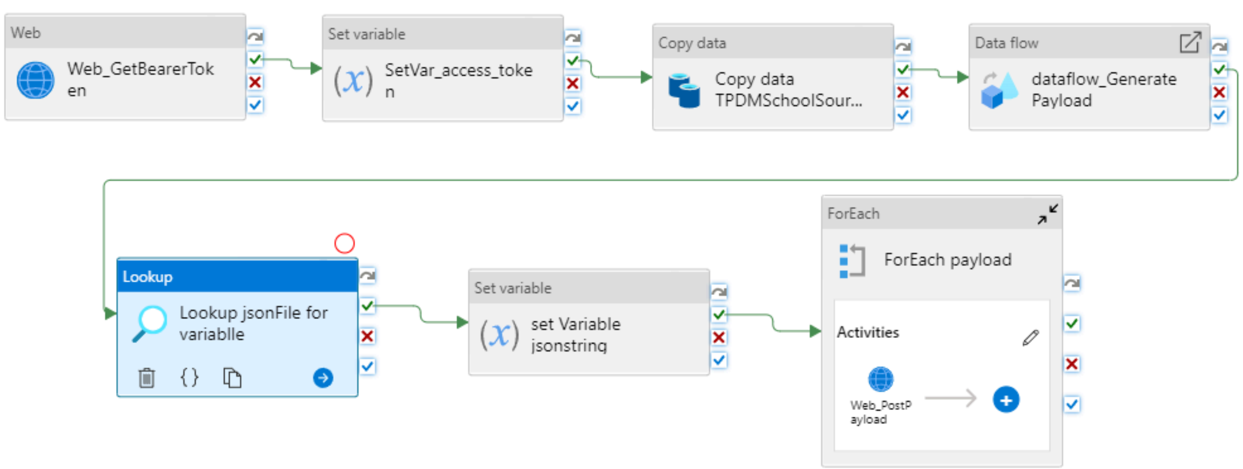
Plan: Data Factory

For the POC an Ed-Fi ODS/API was already set-up on Azure with the scripts provided in the Ed-Fi Exchange. Additionally, the data source file was placed in an Azure VM folder.

Three different methods were tested:

- Loading each JSON payload individually using a ForEach activity.
- Loading all JSON payloads using a Copy activity.
- Loading all JSON payloads directly from a DataFlow.

For the first method this is the pipeline created:



This pipeline uses these activities:

- Web_GetBearerToken: a Web action used to call a custom REST endpoint from an Azure Data Factory or Synapse pipeline. In this pipeline it retrieves the OAuth bearer token from the Ed-Fi API by configuring its settings as follows:
    - URL: Oauth endpoint for the Ed-Fi API (e.g:"https://edfi-test/WebApi/oauth/token")
    - Method: POST
    - Body:

```
{"client_id":"<vendor id>","client_secret":"<vendor
secret>","grant_type":"client_credentials"}
```

    - Headers: a new header was added with the following values:

name	value
Content-Type	application/json

- SetVar_access_token: this Set variable action assigns the access token retrieved by the Web_GetBearerToken action to the access_token pipeline variable with the following variables:
    - Name: access_token
    - Value: @{concat('Bearer ', activity('Web_GetBearerToken').output.access_token)}

This was necessary since the REST sink provided sends “bearer” plus access token which is rejected by the Ed-Fi API due to the lower “b”.

- Copy data TPDMSchoolSource to blob: this Copy activity copies only some columns from the data source file to an Azure Storage account container and adds two hard coded columns by settings its configuration as follows:
    - Source:
        - Source Dataset: TPDMSchoolSource
        - Additional columns:

Name	Value
gradeLevelDescriptor	uri://ed-fi.org/GradeLevelDescriptor#Early Education
educationOrganizationCategoryDescriptor	uri://ed-fi.org/EducationOrganizationCategoryDescriptor#School

    - Sink:
        - Sink dataset: CSVBlobStorage
    - Mapping: these new entries were added:

Source	Type	Destination
SCHID	String	schoolId
SCH_NAME	String	nameOfInstitution
gradeLevelDescriptor	String	gradeLevelDescriptor
educationOrganizationCategoryDescriptor	String	educationOrganizationCategoryDescriptor

- dataflow_GeneratePayload: this dataflow action transforms the CSV data into JSON format with the following settings:
    - Data flow: dataflow_GeneratePayload
    - Compute size: small
- Lookup jsonFile for variable: retrieves the contents of the JsonBlobStorage dataset.
- set Variable jsonstring: sets the value of the jsonString array variable with the contents of the JsonBlobStorage dataset.
- ForEach payload: this activity loops through all the items in the jsonString variable and invokes the Web_PostPayload action for each. The following settings were configured:
    - Items: @variables('jsonsSring')

- Web_PostPayload: another web activity used to post each school payload to the Ed-Fi API schools endpoint. Its settings were configured as follows:
    - URL: Ed-Fi API schools endpoint (e.g: "https://edfi-test/WebApi/data/v3/ed-fi/schools")
    - Method: POST
    - Body: @item()
    - Headers:

name	value
Content-Type	application/json
accept	application/json
Authorization	@variables('access_token')

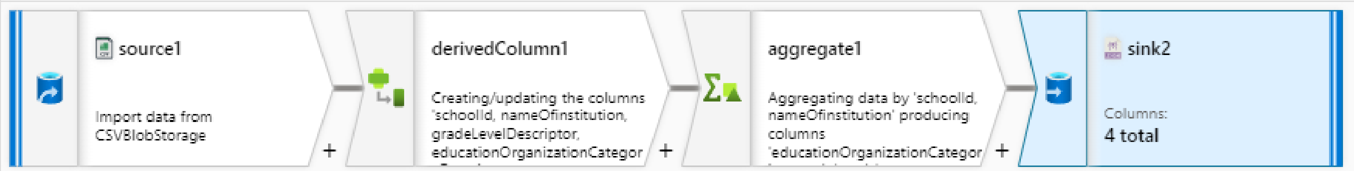
For this pipeline the following variables were created:

Name	Type
access_token	String
jsonsString	Array

Along with two datasets:

- TPDMSchoolSource: a DelimitedText dataset connected thru a Linked Service to the VM hosting the data source file.
- CSVBlobStorage: another DelimitedText dataset connected to an Azure Storage Account container used to stage some of the data source along with hard-coded data.
- JsonBlobStorage: a JSON dataset used to stage the CSVBlobStorage data converted into JSON format.

And finally a dataflow to transform CSV into JSON:


source1
Import data from
CSVBlobStorage
derivedColumn1
Creating/updating the columns
'schoolId, nameOfinstitution,
gradeLevelDescriptor,
educationOrganizationCategor
aggregate1
Aggregating data by 'schoolId,
nameOfinstitution' producing
columns
'educationOrganizationCategor
sink2
Columns:
4 total

This dataflow has these steps:

1. Source1: Imports the data from the CSVBlobStorage dataset by configuring these source settings:
    a. Source Type: Dataset
    b. Dataset: CSVBlobStorage
2. derivedColumn1: creates the gradeLevels and educationOrganizationCategories endpoint fields by adding the following columns to its Derived column's settings:
    a. Columns:

Column	Expression
gradeLevels	@(gradeLevelDescriptor=gradeLevelDescriptor)
educationOrganizationCategories	@(educationOrganizationCategoryDescriptor=educationOrganizationCategoryDescriptor)

3. Aggregate1: aggregates data by schoolId and nameOfInstitution by adding the following columns to:
    a. GroupBy:

Columns	Name As
schoolId	schoolId
nameOfInstitution	nameOfInstitution

    b. Aggregates:

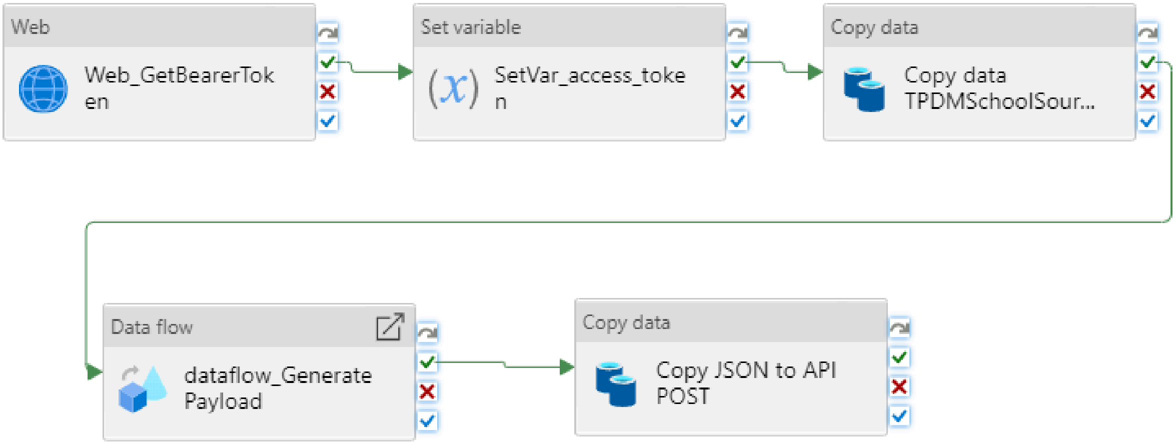
Column	Expression
educationOrganizationCategories	collect(educationOrganizationCategories)

gradeLevels	collect(gradeLevels)

4. sink2: writes the transformed JSON to the JSONStorageBlob dataset.

Unfortunately, this pipeline performance was not good due to the sequential nature of the ForEach activity, leading to an expired token before all the data was loaded.

A second method was tried where the ForEach activity was replaced with a Copy one:



The Copy JSON to API POST Copy data activity was configured as follows:

- Source:
  - Source dataset: JSONBlobStorage
- Sink:
  - Sink Dataset: RestResource
  - Request Method: POST
  - Additional Headers:

name	value
accept	application/json
Authorization	@variables('access_token')

The RestResource REST dataset used by the Copy JSON to API POST activity was configured as follows:

- Connection:
    - Linked Service: RestService

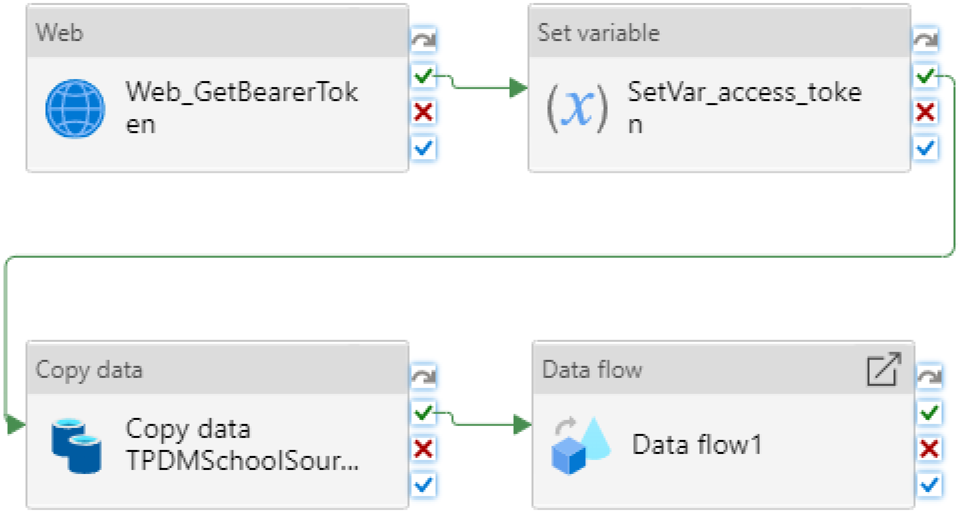
And the RestService used in its sink dataset with the following values:

- Base URL: Ed-Fi API schools endpoint(e.g:"https://edfi-test/WebApi/data/v3/ed-fi/schools")
- Authentication Type: Anonymous

This approach didn't work since the Copy data activity sends an array of school JSON objects to the Ed-Fi API, which replies wit the following error:

```
{\r\n  \"message\": \"Cannot deserialize the current JSON array (e.g. [1,2,3]) into type
'EdFi.Ods.Api.Common.Models.Requests.Schools.EdFi.SchoolPost' because the type requires a JSON object
(e.g. \{\\\"name\\\":\\\"value\\\"}
) to deserialize correctly.\\r
nTo fix this error either change the JSON to a JSON object (e.g. {\\\"name\\\":\\\"value\\\"}) or change the
deserialized type to an array or a type that implements a collection interface (e.g. ICollection, IList) like List<T>
that can be deserialized from a JSON array. JsonArrayAttribute can also be added to the type to force it to
deserialize from a JSON array.\\r
nPath '', line 1, position 1*
```

Finally, the third option was implemented with the following pipeline:



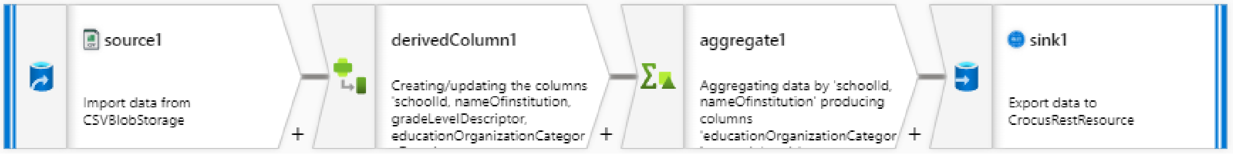
Where the Web_GeatBearerToken, SetVar_access_token and Copy data TPDMSchoolSource to blob actions were copied from the first pipeline and have already been detailed above. The Data flow1 Data flow action settings where configured as follows:

- Data flow: dataflow1

And a new parameter was added to it with these values:

name	value
token	@variables('access_token')

The dataflow1 referenced in the pipeline has the same steps as the dataflow_GeneratePayload described before:



source1

Import data from CSVBlobStorage

derivedColumn1

Creating/updating the columns 'schoolId, nameOfinstitution, gradeLevelDescriptor, educationOrganizationCategor

aggregate1

Aggregating data by 'schoolId, nameOfinstitution' producing columns 'educationOrganizationCategor

sink1

Export data to CrocusRestResource

However, the sink used (sink1) is configured to directly post to the Ed-Fi API by settings its following Sink setting:

- Dataset: RestResource (described previously)

Also, its settings where configured as follows:

- Insert Method: POST
- Format: JSON
- Additional headers:

name	value
accept	application/json
Authorization	$token

And its mapping:

Input Columns	Output Columns

schoolId	schoolId
nameOfInstitution	nameOfInstitution
gradeLevels	gradeLevels
educationOrganizationCategories	educationOrganizationCategories

This pipeline was able to load all the data under an hour with the possibility to decrease that time by using a bigger compute size in the Data flow1 activity's setting.

AWS Glue Studio

This cloud tool has a UI and requires no installation. At the time of writing, the visual option to create a job, didn't have a connector for the Ed-Fi API nor a generic REST connector that could be used and creating one was out of scope for this effort.

However, a POC was created to showcase it as another cloud tool alternative since it provides the option to directly create a Jupyter Notebook.

An Ed-Fi ODS/API was set-up in an EC2 instance (t2.medium) and the NCES csv file was placed in a S3 bucket. The job cluster was configured with 10 G 1X workers .

The following commented code was added to a cell of a new Jupyter Notebook:

```
import requests
import json
import os
from pyspark.sql.functions import *

# This function gets called for each row in the dataframe. It's executed simultaneously by each executor
def postSchool(row):
    try:
        # The actual request done to the Ed-Fi API endpoint. Note we convert the row to a dictionary as the json argument
        res = requests.post(BASE_URL + API_ENDPOINT, verify=False,  headers=api_call_headers, json= row.asDict())
        if res != None and res.status_code == 200:
            print(res.text)
    except Exception as e:
        print("Error: {0}".format(e))

# Set up variables
BASE_URL = "<Ed-Fi API URL (e.g: "https://edfi-test/WebApi")>"
AUTH_ENDPOINT = "/oauth/token"
API_ENDPOINT = "/data/v3/ed-fi/schools"
API_KEY = "<vendor Id>"
API_SECRET = "<vendor secret>"
SOURCE_RAW_PATH = os.path.join("s3://", "<bucket name>", "<NCES file name>")
# EXTRACT: creates a dataframe from the contents of the CSV
source_df = spark.read.format("csv").option("header", "true").load(SOURCE_RAW_PATH)
# TRANSFORM: selects and renames (maps) the SCHID and SCH_NAME columns
df = source_df.select("SCHID","SCH_NAME").toDF("schoolId","nameOfInstitution")
# Adds the hardcoded gradeLevels and educationOrganizationCategories columns
df = df.withColumn("gradeLevels", array(struct(lit("uri://ed-fi.org/GradeLevelDescriptor#Early Education").alias("gradeLevelDescriptor"))))
df = df.withColumn("educationOrganizationCategories", array(struct(lit("uri://ed-fi.org/EducationOrganizationCategoryDescriptor#School").alias("educationOrganizationCategoryDescriptor"))))
# LOAD
data = {"grant_type": "client_credentials"}
# retrieves an authorization token used to post the rows
access_token_response = requests.post(BASE_URL + AUTH_ENDPOINT, data=data, verify=False, allow_redirects=False, auth=(API_KEY, API_SECRET))
# extracts the token from the response
tokens = json.loads(access_token_response.text)
```

```
# builds the header for the post
api_call_headers = {"Authorization": "Bearer " + tokens['access_token']}
# invokes the post function for each row in the dataframe
df.foreach(postSchool)
```

Since the Jupyter Notebook is a functionality provided by Databricks the above code can be run in Azure Databricks and Google Cloud Databricks.

This script loaded the source dataset in 29:27 minutes. Shorter load times can be achieved by scaling up the Ed-Fi API VM and increasing the number of worker nodes in the databricks cluster.

Note that Apache Spark (underlying Databricks technology) allows using SQL language to manipulate dataframes.

Conclusion

All the tools explored in this document are able to concurrently POST data into an Ed-Fi API endpoint Out-Of-Box with both Talend and Nifi being options for on-prem environments and Azure Data Factory and AWS Glue Studio for cloud ones.

For local (on premises) testing, we used a development machine with:

- 8 cores (16 threads)
- 16 Gb RAM
- SSD

Running the Ed-Fi ODS/API v5.3 on IIS and SQL Server 2019.

Tool	Time (~ 100,000 records)	Implementation Time
DIT	32:36 min	2 hrs
Talend	51:16 min	1 hrs
Nifi (1 concurrent thread)	21 min	3 hrs
Nifi (10 concurrent threads)	5 min	3 hrs

The freely available download for Talend only allows for operation in single threaded mode. We were not able to obtain a multi-threaded version in time for this report, but hope to include those results in the future. We expect a multi-threaded version of Talend to perform comparable to NiFi

For testing tools in the cloud we used an Azure instance for the Ed-Fi API:

Standard_DS11_v2 instance

- 2 virtual CPU's
- 14GB Ram
- SSD

Tool	Time (~ 100,000 records)	Implementation Time
Azure Data Factory	51:16 min	2 hrs
AWS Glue Studio (Databricks)	29:26 min	< 1 hr

For Azure Data Factory each POST took between 6-7 seconds to get a response from the Ed-Fi API.. This is probably caused by too few resources being assigned to the API service. With AWS Glue the API responded much quicker (~20ms) but the CPU was spiked at 100%, again pointing to the need for more system resources for the API server. We chose this size machine for the API as this was the size of the instance that the Ed-Fi tools team had used for testing Data Import v1.3 vs the upcoming multi-threaded Data Import v2.0. In our testing the data import tool could take upwards of seven hours to load the same data, system resources are key to any ETL tool running well.

As for ease of setting up and implementing an ETL with these tools we consider Talend to be the easiest, followed by Azure Data Factory and finally by NiFi due to the JOLT transformation used in it.

The DIT is suitable for loading moderate amounts of CSV data where little to no transformations are required. When more complex transformations are needed or multiple data sources need to be combined and the amount of data to load is moderate, Talend is a good choice (even the free open source option). When large amounts of data is to be loaded (despite the transformation or data sources complexity) NiFi would be the best fit. For Azure cloud implementations, Azure Data Factory can handle complex transformation scenarios and high data loads.

As a final consideration, for implementers comfortable with python, scala and SQL languages Databricks is a valid option.

2023 Update

The testing we did on the various products in 2022 was a bit hurried and the hardware used was not consistent between products (i.e. some were completed on development machines rather than standardized environments) in addition the times were from a single attempt. In early 2023 we revisited the testing of some of the products with a sounder methodologies. We removed Talend and AWS Glue and included a new suite of ETL tools: Earthmover and Lightbringer

For these tests, we used the following environments in Azure:

ODS/API - v6.1 running in a Basic (B2) Linux App Service (2 Cores, 3.5 GiB of RAM)

Managed PostgreSQL - Postgres on Azure Database flexible server (Burstable, B2s, 2 vCores, 4 GiB RAM)

ETL tools except Azure Data Factory - Standard DS11 v2 Virtual Machine (2 vCores, 14 GiB memory) Windows (2019 datacenter)

Azure Data Factory used a compute size of medium

As a data source we used the same NCES CCD 2020-2021 school directory as the previous test. Each tool was run 3 times, and then averaged:.

Tool	Avg. Time (~ 100,000 records)
Azure Data Factory	26 min, 45 sec
Data Import v2.0	23 min, 48 sec
Earthmover and Lightbringer	25 min, 48 sec
Nifi	23 min, 8 sec

As shown in the table above all 4 tools are now similar in their performance (including the new tool suite, Earthmover and Lightbringer)